DAT043 – Objektorienterad Programmering

Detta är en exempeltenta som innehåller gamla tentauppgifter av ungefär liknande slag som ni kan förvänta er se på ordinarie tenta i Del 1 respektive Del 2. Dock är inte antalet frågor detsamma som på den riktiga tentan, här har vi bara 5 frågor i Del 1. Ämnena är dessutom på intet sätt heltäckande över allt vad som kan komma på tentan!

Tentamensregler

Den riktiga tentamen består av två delar. För att få **godkänt på tentan (betyg 3) måste man lösa minst fem av sju uppgifter i Del 1** (frågor numrerade 1–7). För att får högre betyg krävs att man utöver detta även löser uppgifter i Del 2 (frågor 8–9). För att få **betyg 4 ska, utöver godkänt på Del 1, även en fråga från Del 2 lösas**. För **betyg 5 skall, utöver godkänt på Del 1, dessutom båda frågorna i del 2 lösas**. Förutsatt att man klarat Del 1 får man självklart testa att lösa båda uppgifterna i Del 2. Lyckas man inte med någon har man ändå säkrat betyg 3, lyckas man bara med den ena så får man betyg 4. Minuspoäng ges ej.

Poäng på Del 2 kan *eventuellt* räknas mot godkänt om det skulle behövas (t.ex. fyra godkända lösningar på Del 1 och en godkänd lösning på Del 2 kan ge en trea i betyg), men det är osannolikt att man klarar frågorna i Del 2 om man inte samtidigt klarat Del 1.

Tillåtna hjälpmedel: Den tresidiga lathund som finns tillgänglig på kurssidan och upptryckt i ett exemplar till varje student vid tentamen (man ska alltså inte ta med sig en egen kopia).

Tentamensgranskning: Efter rättning finns tentamen tillgänglig på expeditionen på plan 4 i EDIT huset och kan granskas där. Önskar man diskutera rättningen kommer man kunna boka tid för detta genom att fylla i det formulär som kommer finnas länkat till från kurshemsidan efter att tentan är rättad. Notera att tentan i så fall ska lämnas kvar på expeditionen.

- Implementeringar ska skrivas i Java. Oväsentliga syntax- och namnfel eller liknande, betyder inte att lösningen underkänns. Det viktiga är att lösningar bedöms vara tillfredställande, d.v.s. att studenten med sitt svar tydligt visar förståelse för problemet i uppgiften och dess lösning.
- Skriv tydligt och välstrukturerat. Svårförståeliga lösningar kan underkännas.
- Lösningar som är onödigt krångliga eller inte följer god programmeringsstil, dels allmänt och dels med tanke på idéerna med objektorienterad programmering, kan underkännas.
- Om det inte uttryckligen står motsatsen i uppgiften kan du använda klasser och metoder i Java:s API.
- Om det inte uttryckligen står motsatsen i uppgiften får du definiera egna hjälpmetoder.
  Importeringar av klasser i Java:s API behöver inte skrivas ut.

DEL 1

Uppgift 1

Du kan anta att klassen Person finns, är felfri och är synlig. Vilket påstående är då korrekt om klassen C som är definierad nedan? Om du svarar i, ii eller iii krävs att du motiverar ditt svar!

```
import java.util.*;

   public class C {
     private LinkedList<Person> list;
     private String name;
     public C(String name) { this.name = name; }
     public void add(Person p) { list.add(p); }
}
```

i. Exekveringen riskerar att avbrytas om object av klassen används.
ii. Klassen går inte att kompilera
iii. Det går inte att skapa objekt av klassen.
iv. Inget av ovanstående problem.

Uppgift 2

I ett program har man lagt följande rader:

```
   int x = 8;
   String sa = "AB8";
   String sb = sa;
   String sc = "AB" + x;
   if (sa==sc) {
     System.out.println("lika");
   } else {
     System.out.println("olika");
   }
   if (sa==sb) {
     System.out.println("lika");
   } else {
     System.out.println("olika");
}
```

Ange vilken utskrift programmet ger och förklara varför.

Uppgift 3

Implementera metoden

```
public static boolean isIncreasing(int[] a)
```

som returnerar true om a innehåller en växande talföljd, d.v.s. om varje tal i `a` är minst lika stort som föregående tal. Om `a` inte innehåller en växande talföljd så ska metoden returnera false. Elementen i `a` ska inte ändras av metoden.

Uppgift 4

Skriv en klass Student som representerar en student. Varje student har ett namn av typ String och ett student-ID som är en integer. Klassen ska ha en konstruktor som tar en sträng och en integer som argument och sätter studentens namn till strängen och student ID till integern. Instansvariablerna ska *inte gå att ändra utifrån*, så använd en lämplig tillgänglighetsmodifierare. Skriv även metoder som man *kan komma åt utanför klassen* som returnerar namn respektive ID.

Uppgift 5

Givet följande gränssnitt och klasser:

```
interface A {
}

public class B implements A {
}

public class C extends B {
}
```

Vilka rader i följande kodutsnitt innehåller fel?

```
A x1 = new A();
B x2 = new B();
C x3 = new B();
B x4 = new C();
A x5 = new C();
A x6 = x2;
B x7 = x5;
C x8 = (C)x2;
C x9 = (C)x4;
```

Ange vilken eller vilka rader som innehåller fel och för varje felaktig rad om felet uppstår vid kompilering eller exekvering av programmet. Använd numret på variabeln som deklareras när du anger rader, d.v.s. ett tal mellan 1 och 9.

DEL 2

Uppgift 6

Implementera två generiska statiska metoder för att beräkna unionen och snittet av mängder. Metoden union ska ta två argument, a och b, av typen Set<E>, där Set är Java collection frameworks interface för mängder och E är en typparameter. Metoden ska skapa och returnera en ny mängd (ett objekt av typen Set<E>) som innehåller alla element som finns antingen i a eller b. De två ursprungsmängderna, a och b, ska vara oförändrade.

Metoden intersection ska ha samma signatur men istället för unionen istället skapa en mängd som utgör snittet, d.v.s. innehåller alla element som finns både i a och b.

Du ska definiera dessa två metoder inklusive signaturen d.v.s. det som står före metodkroppen (det inom { }).

Du kan använda implementeringen HashSet när du skapar den nya mängden.

Tips: Interfacet Set utökar Collection som utökar Iterable.

Uppgift 7

Skriv ett program Find som läser en befintlig textfil och söker efter de rader i filen som innehåller en viss sträng. De funna raderna skall kopieras till en ny fil. Indata till programmet skall ges som argument på kommandoraden. (Programmet skall alltså inte läsa indata från System.in.) När man startar programmet skall man på kommandoraden ge följande tre argument: namnet på den befintliga filen, namnet på den nya filen samt den text man söker. Man kan t.ex. ge kommandot

```
java Find bilar.txt volvo.txt Volvo
```

Här kommer alla de rader i filen bilar.txt vilka innehåller texten Volvo att kopieras till filen volvo.txt. Programmet skall kontrollera att antalet argument är korrekt och att filerna går att öppna. Om något skulle vara fel skall en felutskrift ges och programmet avslutas.

LÖSNINGAR

1)
Alternativ (i). Listan initieras implicit till null, så om metoden add anropas sker ett undantag.

2)
```
olika
lika
```
==-operatorn jämför för referenstyper referenserna, d.v.s. om operanderna pekar på samma objekt. sa och sc innehåller samma sträng men är ej samma objekt. sa och sb pekar på samma objekt.

3)
```
public static boolean isIncreasing(int[] a) {
  for (int i = 0; i < a.length-1; i++) {
    if (a[i+1] < a[i]) return false;
  }
  return true;
}
```

4)
```
public class Student{
     private int id;
     private String name;

     public Student(String name, int id){
          this.name = name;
          this.id = id;
     }

     public String getName(){return name;}
     public int getID(){return id;}
}
```

5)
Kompileringsfel på rad 1,3 och 7. Exekveringsfel på rad 8.

6)
```
public static <E> Set<E> union(Set<E> a, Set<E> b) { Set<E> c
= new HashSet<E>();
Iterator<E> i = a.iterator();
while (i.hasNext()) {
      c.add(i.next());
    }
    i = b.iterator();
    while (i.hasNext()) {
      c.add(i.next());
    }
return c; }
  public static <E> Set<E> intersection(Set<E> a, Set<E> b) {
```

```
    Set<E> c = new HashSet<E>();
    Iterator<E> i = a.iterator();
    while (i.hasNext()) {
      E e = i.next();
      if (b.contains(e)) c.add(e);
    }
return c; }
```

7)

```
public class Find {
}
public static void main(String[] arg) {
  if (arg.length != 3) {
    System.out.println("Felaktigt antal argument");
    System.exit(0);
  }
  Scanner s = null;
  try {
    s = new Scanner(new File(arg[0]));
  }
  catch (IOException e) {
    System.out.println("Kan inte öppna filen " + arg[0]);
    System.exit(0);
  }
  PrintStream p = null;
  try {
    p = new PrintStream(new File(arg[1]));
  }
  catch (IOException e) {
    System.out.println("Kan inte skapa filen " + arg[1]);
    System.exit(0);
  }
  while (s.hasNextLine()) {
    String line = s.nextLine();
    if (line.indexOf(arg[2]) >= 0) p.println(line);
  }
s.close();
p.close(); }
```